CS 166 Homework 1

Leo Martel (lmartel)

4/9/2014

Question 1 (Sparse Tables with O(1) Queries)**.** *Design a data structure to compute largest k for* $2^k \leq j-i+1$ *in constant time for use in RMQ.*

Overview: we simply precompute k such that $2^k \leq i < 2^{k+1}$ for $1 \leq i \leq n$ and store the values in an array.

Preprocessing: Initialize an array A of length n+1 (allowing one-indexing for simplicity).

Starting with $k = 0, p = 2^k = 1, x = 1$. Repeat ([If $2p \leq i$ then increment k and set $p = 2p$.] Write $A[x] = k$. Increment x.) until the array is filled in, and $x = n + 1$.

Each step gives us the correct k, because we start with correct $k = 0$ for $x = 1$, and by induction if k is correct for $x - 1$, either $k + 1$ is correct for x (which we check) or k is correct for x, which we fall back to. This is because adding 1 never moves us past more than one power of two.

Each step takes $O(1)$ time, doing only multiplication, an array write and a few increments. We do n steps, since we increment x each time from 1 to n. So the total preprocessing work is $O(n)$

Querying: $A[j - i + 1] = k$ gives us the largest k such that $2^k \leq j - i + 1$, as shown in the previous step.

The minimum range $j - i + 1 = 1$ and the maximum range is n, so all possible valid queries can be answered by the array.

A query requires some arithmetic and an array access, which takes $O(1)$ time.

Question 2 (Area Minimum Queries)**.**

i. An $\langle O(mn), O(min(m, n))\rangle$ data structure for AMQ.

ii. An $\langle O(mn \lg m \lg n), O(1)\rangle$ data structure for AMQ.

Question 3 (Hybrid RMQ Structures)**.**

i. Lemma: for any fixed $k \geq 1$, there is an RMQ data structure with time complexity $\langle O(n \lg^{(k)} n), O(1)\rangle$ called D_k.

Proof: by induction on k.

Base case: $k = 1$. A sparse table gives us time complexity $\langle O(n \lg n), O(1)\rangle$, proved in class.

Inductive step: given D_k, can we build D_{k+1}?

We use the hybrid approach with D_k as both our top and bottom RMQ structures, with a block size of log n.

Preprocessing time: $(p_1 = p_2 = O(n \lg^{(k)} n)))$

$$O(n + p_1(n/b) + (n/b)p_2(b))$$

We have $p_1(n/b) = (n/\lg n) \lg^{(k)}(n/\lg n) \leq (n/\lg n) \lg^{(k)} n$ (since $(n/\lg n) \leq n$ for sufficiently large n)

And then $(n/\lg n) \lg^{(k)} n \leq (n/\lg n) \lg n = n$ (since $\lg^{(k)} n \leq \lg n$) so $p_1(n/b) = O(n)$.

$$= O(n + n + (n/\lg n)b \lg^{(k)} b)$$

$$= O(2n + (n/\lg n) \lg n \lg^{(k)} \lg n)$$

$$= O(n \lg^{(k+1)} n)$$

Query time: $(q_1 = q_2 = O(1))$

$$O(q_1(n/b) + q_2(b))$$
$$= O(1)$$

So this hybrid approach is D_{k+1}, completing the induction.

ii. Query times increase because the constant factor increases. The asymptotic runtime is $O(1)$ because the runtime does not depend on n, only on k–the number of logs we're taking–which is not related to the size of the input. The source of the runtime increase is that each level of hybridization requires us to build two RMQ structures for the top and the bottom, which are hybrids themselves, and so on.

Question 4 (Implementation)**.** *Turned in electronically.*